```
/* @(#)e_exp.c 5.1 93/09/24 */
/*
 * ====================================================
 * Copyright (C) 1993 by Sun Microsystems, Inc. All rights reserved.
 *
 * Developed at SunPro, a Sun Microsystems, Inc. business.
 * Permission to use, copy, modify, and distribute this
 * software is freely granted, provided that this notice
 * is preserved.
 * ====================================================
 */

/* exp(x)
 * Returns the exponential of x.
 *
 * Method
 *   1. Argument reduction:
 *      Reduce x to an r so that |r| <= 0.5*ln2 ~ 0.34658.
 *      Given x, find r and integer k such that
 *
 *               x = k*ln2 + r,  |r| <= 0.5*ln2.
 *
 *      Here r will be represented as r = hi-lo for better
 *      accuracy.
 *
 *   2. Approximation of exp(r) by a special rational function on
 *      the interval [0,0.34658]:
 *      Write
 *          R(r**2) = r*(exp(r)+1)/(exp(r)-1) = 2 + r*r/6 - r**4/360 + ...
 *      We use a special Remes algorithm on [0,0.34658] to generate
 *      a polynomial of degree 5 to approximate R. The maximum error
 *      of this polynomial approximation is bounded by 2**-59. In
 *      other words,
 *          R(z) ~ 2.0 + P1*z + P2*z**2 + P3*z**3 + P4*z**4 + P5*z**5
 *      (where z=r*r, and the values of P1 to P5 are listed below)
 *      and
 *          |                  5          |     -59
 *          | 2.0+P1*z+...+P5*z   -  R(z) | <= 2
 *          |                             |
 *      The computation of exp(r) thus becomes
 *                             2*r
 *              exp(r) = 1 + -------
 *                            R - r
 *                                 r*R1(r)
 *                     = 1 + r + ----------- (for better accuracy)
 *                                2 - R1(r)
 *      where
 *                               2       4             10
 *              R1(r) = r - (P1*r  + P2*r  + ... + P5*r   ).
 *
 *   3. Scale back to obtain exp(x):
 *      From step 1, we have
 *         exp(x) = 2^k * exp(r)
 *
 * Special cases:
 *      exp(INF) is INF, exp(NaN) is NaN;
 *      exp(-INF) is 0, and
 *      for finite argument, only exp(0)=1 is exact.
 *
 * Accuracy:
```

```
	    }

    /* argument reduction */
	if(hx > 0x3fd62e42) {		/* if  |x| > 0.5 ln2 */
	    if(hx < 0x3FF0A2B2) {	/* and |x| < 1.5 ln2 */
		hi = x-ln2HI[xsb]; lo=ln2LO[xsb]; k = 1-xsb-xsb;
	    } else {
		k  = invln2*x+halF[xsb];
		t  = k;
		hi = x - t*ln2HI[0];	/* t*ln2HI is exact here */
		lo = t*ln2LO[0];
	    }
	    x  = hi - lo;
	}
	else if(hx < 0x3e300000)  {	/* when |x|<2**-28 */
	    if(huge+x>one) return one+x;/* trigger inexact */
	}
	else k = 0;

    /* x is now in primary range */
	t  = x*x;
	c  = x - t*(P1+t*(P2+t*(P3+t*(P4+t*P5))));
	if(k==0)	return one-((x*c)/(c-2.0)-x);
	else		y = one-((lo-(x*c)/(2.0-c))-hi);
	if(k >= -1021) {
	    u_int32_t hy;
	    GET_HIGH_WORD(hy,y);
	    SET_HIGH_WORD(y,hy+(k<<20));	/* add k to y's exponent */
	    return y;
	} else {
	    u_int32_t hy;
	    GET_HIGH_WORD(hy,y);
	    SET_HIGH_WORD(y,hy+((k+1000)<<20)); /* add k to y's exponent */
	    return y*twom1000;
	}
}
DEF_STD(exp);
LDBL_MAYBE_CLONE(exp);
```